Analysing software using deep learning

Mohammad Wazed Ali

August 27, 2018

Abstract

Analyzing software using deep learning project has been implemented as part of masters curriculum at TU Darmstadt under the course title "Analyzing software using deep learning".In this project an attempt has been made to predict the arbitrary number of missing tokens in a JavaScript program. Two neural network has been implemented to complete the task e.g. feed forward network (FFN) and Recurrent neural network (RNN). Corpus of 800 JavaScript program has been taken as training dataset. These programs then tokenized as sequence of tokens making arbitrary number of holes (max 3) within them at arbitrary positions.These sequence of tokens was feed to the neural network to train it. Another corpus of 200 JavaScript program has been taken for testing purpose. They also has been tokenized in same way and made arbitrary hole within them. These sequence of tokens then feed to the network as prefix and suffix of the hole to predict the tokens in hole. The FFN could predict the missing tokens with an accuracy of around 65% while the maximum number of missing token in the hole was one, with an accuracy of around 45% while the maximum number of missing token in the hole was 2 and 25% while the maximum number of missing token in the hole was 3. The result for the LSTM network was little different than FFN. It was able to predict with an accuracy of around 70% while the maximum number of missing token in the hole was 1 and 55% while the maximum number of missing token was 2 and 35% while the maximum number of missing token was 3. The over all accuracy could be improved with more data set.

1 Introduction

An Artificial Neural Network (ANN) is based on a collection of connected units or nodes called artificial neurons which loosely model the neurons in a biological brain. Despite having various type of Neural Network only FFN and RNN has been considered for this project. In FFN the weights and biases are initialized with arbitrary number using some initializer function. Then these weights and biases are adjusted with each feed of the training samples with an objective to reduce the cost function. Various activation and cost function can be used depending on the purpose of the network. This network does not consider the outcome of the previous step of the training while adjusting the weights and biases. The RNN also works in the same concept except it considers the outcome of the previous step while adjusting the weights and biases. As result the impact of the sequence data is preserved in this type of network. For the implementation of this project Long Short Term Memory cell a special type of RNN has been used.

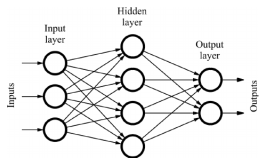

Figure 1: FFN

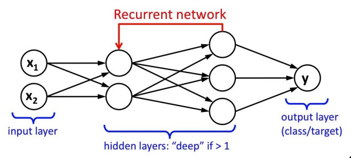

Figure 2: RNN

2 Project setup

Python with tensor flow has been considered for the purpose of this project. Python virtual environment has been setup with the following library:

a. Python 3.5
b. Tensorflow 1.1 (with CPU support only)
c. TFLearn 0.3.1

The provided framework has been used to implement the project. The framework contains two file namely runner.py and code_completion_baseline.py. runner.py file was used to generate the data for training and evaluation from the provided corpus of files whereas code_completion_baseline.py was used to prepare the data, create the network, feed the training data and predict the missing tokens. All 800 files provided for the project was used to train the network. Prediction accuracy has been calculated as per the following formula:

$$accuracy = \frac{nb.of correct prediction}{total nb.of queries}$$

code_completion_baseline.py has been rewritten to improve the accuracy of the network. It was renamed as code_completion_improved.py for FFN and code_completion_final.py for LSTM network.

3 Approach towards solution

Several intentional flaws were integrated in the code_completion_baseline.py e.g. considering single token from the prefix, ignoring the suffix, predicting exactly one token always. Primary goal to solve the problem was to improve the code_completion_baseline.py by mitigating these flaws first, then try to improve more by adjusting the hyper parameters. Two different approach has been attempted to solve the problem. Details of the approaches attempted are following:

3.1 FFN

To solve the problem with FFN a sequence of 10 tokens has been taken as training sample from the whole corpus of tokens making either one or two holes within them. Then these samples has been feed to the network in a batch of 2000 as a n_hot vector. Four layer has been considered for this network. Input layer with dimension of (number of unique tokens X length of training samples), two fully connected hidden layer with rectified linear unit as activation function and 250 neurons in each layer and output layer using softmax as activation function. Total training samples was 960000 (due to limitation of memory and computing power of PC). Analyzing the empirical result of test data from this network, it was found that this network could predict 60% to 65% tokens correctly while max hole size was one, 40% to 45% tokens correctly while max hole size was two and 20% to 30% tokens accurately while max hole size was set to three. Random output for 1,2 and 3 max missing hole is shown in listing 1,2 and 3. The network was also trained changing the hyper parameter but output was best with this parameter. One of the major limitation of this network was it doesn't consider the previous occurrences of tokens while predicting the current one i.e. no back propagation to adjust the weights and biases.

```
Unique tokens: 86
x,y pairs: 960000
Accuracy: 124 correct vs. 76 incorrect
    = 0.62
```

Listing 1: Max 1 hole accuracy in FFN

```
Unique tokens: 86
x,y pairs: 960000
Accuracy: 82 correct vs. 118 incorrect
    = 0.41
```

Listing 2: Max 2 hole accuracy in FFN

```
Unique tokens: 86
x,y pairs: 960000
Accuracy: 52 correct vs. 148 incorrect
    = 0.26
```

Listing 3: Max 3 hole accuracy in FFN

3.2 RNN

To solve the problem of FFN (no back propagation) a second approach has been attempted using RNN (LSTM) network. In this approach time stamp of sequence has been preserved. Feed data was prepared from the token_lists with a length of 6 token creating a hole of one or two tokens. 4 token from the prefix of hole and 2 token from the suffix of hole. They have been concatenated as suffix[1] + suffix[0] + prefix[3] + prefix[2] + prefix[1] + prefix[0] with a hope of better performance considering prefix will have more impact on the out come than suffix . Data has been feed to the network as sequence of one_hot vector of each token appearing in the sequence. As the input dataset was huge computing power of the machine was a limiting factor to train the network on all possible sequences. Instead 160000 training samples was used to train the network. By analyzing the empirical result it has been found that this network could predict 65% to 75% token accurately while number of max hole size was 1, 45% to 50% while max hole size was 2 and 35% to 45% while max hole size was 3. The network was also trained with other hyper parameter but no significant improvement was found. Listing 4,5 and 6 shows random prediction for each hole size.

```
Unique tokens: 86
x,y pairs: 240000
Accuracy: 137 correct vs. 63 incorrect
    = 0.685
```

Listing 4: Max 1 hole accuracy in LSTM

```
Unique tokens: 86
x,y pairs: 240000
Accuracy: 103 correct vs. 97 incorrect
    = 0.515
```

Listing 5: Max 2 hole accuracy in LSTM

```
Unique tokens: 86
x,y pairs: 240000
Accuracy: 86 correct vs. 114 incorrect
    = 0.43
```

Listing 6: Max 3 hole accuracy in LSTM

4 Analyzing the result

From empirical result of both network it was obvious that accuracy of LSTM will be better than FFN as LSTM consider the weights and biases from previous sequences to adjust the weights and biases of current sequence (back propagation). For example, while training with same training samples (160000) accuracy of LSTM was always around 10% higher than FFN in all three cases (1, 2 and 3 holes). Listing 7 shows a random accurate prediction of 3 missing tokens hole.

```
expected:
[{'type': 'Punctuator', 'value': '}'},
    {'type': 'Punctuator', 'value':
    ')'}, {'type': 'Punctuator',
    'value': ';'}]
predicted:
[{'type': 'Punctuator', 'value': '}'},
    {'type': 'Punctuator', 'value':
    ')'}, {'type': 'Punctuator',
    'value': ';'}]
------------- correct ----------------
```

Listing 7: correct prediction of 3 missing tokens

The overall accuracy could be improve more with larger training samples. Another facet of accuracy was unknown number of tokens in holes. As the number of tokens in the hole was unknown so a complete sequence was considered when the predicted token was matched with first token of the suffix. This approach also reduced the accuracy as sometimes it gave false negative accuracy about the length of the missing tokens. In many cases it was found that the model could predict 1/2 missing tokens accurately but could not predict the next token (first token of the suffix) thus producing inaccurate results. Listing 8 shows one such example. Here though the network could predict the missing token correctly but the out come was incorrect. Overall accuracy could be improved if number of missing tokens in each hole could be known.

```
expected:
[{'type': 'Punctuator', 'value': ','}]
predicted:
[{'type': 'Punctuator', 'value': ','},
    {'type': 'Identifier', 'value':
    'ID'}, {'type': 'Punctuator',
    'value': ','}]
------------- incorrect --------------
```

Listing 8: False incorrect prediction

5 Conclusion

Despite limitation of computing power and less number of training samples the implemented network could predict a good number of missing tokens correctly. Though with increasing numbers of missing tokens it became more difficult to predict the correct sequence of tokens as any single false prediction in the sequence will result an incorrect prediction. Also this is not a very appropriate idea to check for the correctness of the prediction basing on the first token in the suffix. Number of data file is not sufficient to get the most accuracy. Accuracy could be increased more with more training samples and knowing the number of missing tokens in each hole.

References

[1] Wikipedia,
https://en.wikipedia.org/wiki/Artificial_neural_network

[2] Udemy,
https://www.udemy.com/